Zcash & Zerocash: Data mining, cryptanalysis and key reuse

Killian Davitt, Lena Petersen

March 25, 2018

Intro

This document was written as part of the GA18 Cryptanalysis course to explain my work as part of Project PK. We will discuss Zerocash, Zcash, The Zcash client, How we mined interpreted Zcash data and finally how we searched for duplicated random values in ECDSA signatures and reused public keys.

Zerocash

Zerocash [1] is a cryptocurrency template. While mostly being similar to bitcoin, it both attempts to provide facility for highly anonymous transactions in cryptocurrency. We will not discuss Zerocash much here, instead we will focus on Zcash, it's most prominent implementation. The prominent goal of both however is to facilitate highly anonymous cryptocurrency transactions using zero knowledge proofs.

Zcash

Zcash [2] is one particular instance of the Zerocash protocol. It is mostly similar to the Zerocash protocol as described in [1] but has some slight but significant differences [3]. Zcash currency is denoted as *ZEC*.

Before discussing the format of anonymous transactions, it is worth noting that Zcash also includes the "Equihash" hash function as part of it's proof of work which is designed to use a *memory hard* problem. The equihash function requires a certain amount of RAM to compute and thus it is expected to be far more difficult to run the equihash function on large scale ASICs which is why it was selected for Zcash.

One distinct problem with Zerocash is the time it takes to generate the zero knowledge proofs needed for highly anonymous transactions. It can take up to 1 minute to generate an anonymous transaction. Thus Zcash in it's implementation of Zerocash allows for both highly anonymous transactions as well as standard transactions similar to bitcoin which do not use zero knowledge proofs. If a user does not have the time to wait to create a zero knowledge proof they can simply opt to create a standard non-anonymous transaction. The two types of transactions are known as "Shielded" and "Un-Shielded" and are interoperable; i.e. A user can receive funds through an unshielded transaction and then immediately send those same funds in a shielded transaction to someone else.

Structure of transactions

Firstly we will focus on unshielded transactions as shielded transactions are quite complex and will require a great deal of effort to explain.

Unshielded Transactions

The following image shows an example of an unshielded transaction in JSON. The structure is quite similar to a bitcoin transaction.

The inputs being consumed are described in "Vin" and the outputs of the transaction are described in "Vout". The empty field "VJoinSplit" is populated in shielded transactions only.


```
{
  "txid": "fcd70449e4daa2283406b4c09fbc72c415367deccf9a5e459852e6ecbdc9f47d",
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "bf48024235d408870377cdeba626842593bd9f64d56af5594316f8bf9de392fe",
      "vout": 1,
      "scriptSig": {
        "asm": "3045022100f99d2709e18a7f113d1642a1f59138aa2f42aa51937ba0edfaabf3d9f74b6096022059f10bd89ab2e3f0303ad1168743cc679b8b90474e64892d481ffc056b3d415301 020fc88d194461ae5925abc94f64857b8ae2abf8aa52b77dec844c703377c7df53",
        "hex": "483045022100f99d2709e18a7f113d1642a1f59138aa2f42aa51937ba0edfaabf3d9f74b6096022059f10bd89ab2e3f0303ad1168743cc679b8b90474e64892d481ffc056b3d41530121020fc88d194461ae5925abc94f64857b8ae2abf8aa52b77dec844c703377c7df53"
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01246000,
      "valueZat": 1246000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 a11093f9b2210ca7dd05b3fb459bd635ee6721dd OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a914a11093f9b2210ca7dd05b3fb459bd635ee6721dd88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "t1YZEf5hoPqetfmZ7TLEJSqvuWL9k1FZuk8"
        ]
      }
    },
  ],
  "vjoinsplit": [
  ]
}
```


Figure 1: Example of an unshielded tx in json

Aside from n Vin blocks and m Vout blocks, a unshielded transaction also contains the transaction id, version and locktime.

1. **Vin:** Quite similarly to bitcoin, vin's describe the source of the funds being spend in the current transaction, more specifically, the transactions which output the funds being now spent. Moreover a vin describing sourced funds also includes an ECDSA signature on the script of the transaction
2. **Vout:** Each Vout included in a transaction essentially describes one outputted amount which may be spent. That is, a Vout describes a way of spending some amount of the source funds.

Before we discuss shielded transactions there are two cryptographic components we need to briefly consider.

Zero Knowledge Commitments

Zero Knowledge commitments are the method used by Zcash to show ownership of coins being spent. In a sense, when shielded amounts are created, a random values is "committed to". In unshielded transactions, users verify their ownership of an amount by signing the input transaction. When attempting a private transaction a signature protocol does not work since doing so would inherently be less than perfectly anonymous. Instead, a commitment scheme is used, and a user proving their ability to spend an amount comes from them revealing the opening value of the commitment.

zk-SNARK

zk-SNARK is a particular zero knowledge proof system which is used in Zerocash & Zcash to allow a user to prove their ownership of a coin without revealing who they are. Consider traditional bitcoin where the spender of a coin must sign a transaction to prove they had the authority to spend the coins, creating a signature necessitates including the public key of the user which of course can be used to deanonymise the user. The zk-SNARK proof system is used here in conjunction with the zero knowledge commitment scheme to allow users to prove they have the authority to spend coins without revealing their identity. We will not consider any cryptanalysis on the zero knowledge proof scheme as it is much too complex and is beyond our scope..

zk-SNARK stands for Zero Knowledge Succinct Non-interactive Argument of Knowledge.

In particular, Zcash uses the "lib-snark" C++ implementation of this zero knowledge proving scheme.

Shielded Transactions

Shielded transactions are significantly more complex than unshielded ones. Shielded transactions are often referred to as "JoinSplits". A shielded transaction can be one of three formats either

- Transferring money from shielded inputs to shielded outputs. (In this case there is no information about the amounts revealed, the transaction could be for 10ZEC or 10000ZEC we have no idea)

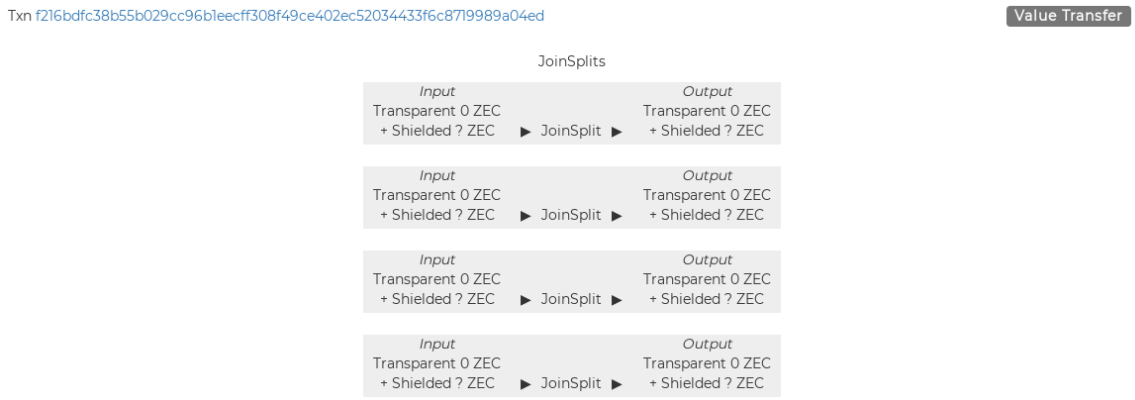

Figure 2: Example of (shielded → shielded) tx from explorer.zcha.in

- Transferring money from unshielded inputs to shielded outputs. (In this case we can see how much money is consumed by the transaction but we don't know to which outputs it went

Figure 3: Example of (unshielded → shielded) tx from explorer.zcha.in

- Transferring money from shielded inputs to unshielded outputs. In this case we cannot see where the funds are coming from but we can see to which address the funds are sent to.

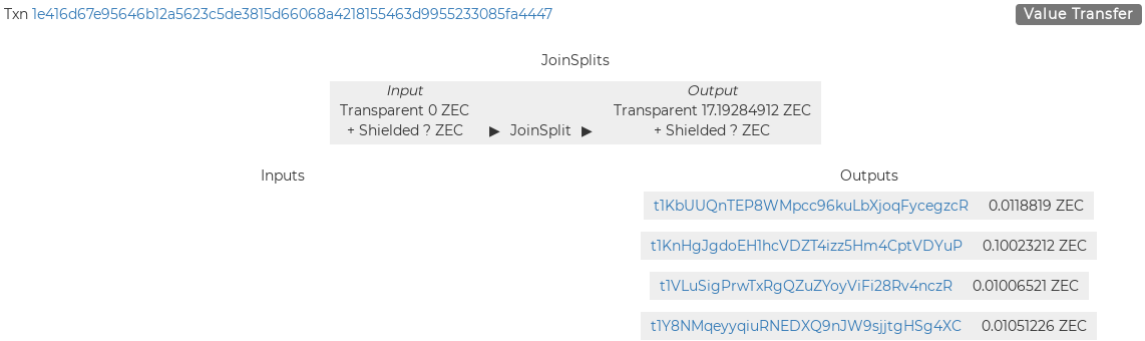

Figure 4: Example of (shielded → unshielded) tx from explorer.zcha.in

Precise format of shielded transactions


```
{
  "txid": "ac8c02e26f0c01f72113920666c77cf357a649ab60d6dc2e00315e44e3e982c5",
  "version": 2,
  "locktime": 0,
  "vin": [
  ],
  "vout": [
    {
      "value": 0.01000000,
      "valueZat": 1000000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 6f890440f77df21db7a7bdbda03355ffed52b97a OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9146f890440f77df21db7a7bdbda03355ffed52b97a88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "t1U3M9i9DJphYwCt7ThCSxSP8B8xvvohyXP"
        ]
      }
    },
    {
      "value": 1530.13004200,
      "valueZat": 153013004200,
      "n": 12,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 b26393a80470de91acaf8eb0e06235398c2755e1 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a914b26393a80470de91acaf8eb0e06235398c2755e188ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "t1a8qaCQkpVjTvFpxjna6P5fXknmMYecNJr"
        ]
      }
    }
  ],
  "vjoinsplit": [
    {
      "vpub_old": 0.00000000,
      "vpub_new": 1001.54105000,
      "anchor": "7aa32f3352fbe74eceeb5a4ba2bc3a3172f127ef0e6ba22667fcc7fc284af135",
      "nullifiers": [
        "f2236193145a6f320a44290d1a78b0b32f7c733066a3bb6c8ed4dad32c8a915c",
        "c235a9adfc9cc0518deaf22752f92ce6146c327e182b39fddcc72cc1ecdb94dd"
      ],
      "commitments": [
        "b1698a4d7acf6534fa0fc1cc86f6a39a23d553349e827cbe60c8b85c75a48ede",
        "fbde69ccfc990d72ea942a8b6adc7016089385c2e1b0f836641e14fa6fcda9de"
      ],
      "onetimePubKey": "6a5653541cd0e9238dafc3734e287733ae94ecc5589d779648529357b1557565",
      "randomSeed": "86127029d181bc31f3af00c11f2ac497497f4312bf6df74c3d002ad1d64734c6",
      "macs": [
        "5873107b3a0781593d20a07a9dd2e510dc814eb14a4e9450f6b6ae0bbe9229f1",
        "2b2082b1f29542f0f60cf8905451cdffa0da1274d08168220319956240e242e8"
      ],
      "proof": "021fb5c1092ec5e7a11bac9b38a82d59b3563009c4584b73f1cfdf03a6e4a1433003284a2d31d0e49dab65b4a7a95221088da6f644ff94cf178446f0501c613717f80b075b6be41f77f15bd8393
408ec9643acb517dd577880c212850e0d238a5f5e4b21ae8e34a02cbf6af1643d3b4f6dd192ca7645f0ca87af14bc4b2146109aabc2030c55e1e8760520b4fe9fd1fc9e3334a6ec78be5a5ed15fc829583e4b46d755
fe0300a1cfa13ac638e1ed0c385651c29c63373fe12bba4a943a28ae1b8bd2f6ef39031cdc27e7a048a4779365a9fe52e006950e589579478e3df56b4ef86e247f757f021921e6acd73858d8ab39dcb2e28b95babe5
514a83835150392b79b7a6d5abe19032789e504876b00d07d23bc4e3eba27cde85fad88368677b4ce03ae6348883923",
      "ciphertexts": [
        "363050b33fb4e6efae5c5c528373ea76e89f42af9eae8276dcb295a1fca389bb7f44dc37b2e3f303a2b125306fc889f014de36a46d125bab78d3d689645ef12d4f6f781f3c4e0cef039dc2f0eb3bce2e3a
a356cad1858a70836e8105654a96247dc6da5cc14684a4cc1587ed014ae14739d660a96fb8c61b440cd343a421d394ab52a546e289f7f455e88f8566a1d780819eba006a529c5c2e3f32e568f6e9331a889160a5531
5b728448422f28cd9d0d24721bb0bc162a1f71c44521b1b95467b536f50cc66624154d9522f732a38b05ee1be0f1bb1384271d2866c241cfb8332462f09c98b6933a87f208785b15e143eb3a0a8a85dba13501d43bb
24454440fd2a3aa994d14af8575baf2607280c5d6cbdee71eb0bbe90439344731f67b3f96984b1411ad289f19c79f8b0edffe8774d355e20b73be7995d1684732dd478fe3beda9cac0032509dc8e47630b88565948a
60feceacb8190540d273dc9bdb19b4cbbd7eabcbf6a400cef645e02378e1b4fcf6d7a728f03a9cf0b263de611245d6ce465a726f7baae51c2e2e45482d2d2eb18221d9a41c234ce3655f8be35d590a7c448b840acb0
14f445111b3716df0084af1e36f03b84e6f41f20c0ad3612ded2fb5e5f9b62db7615aa36b25cd67d80bc4e54fc138bf126cc6af4948df41fc3916f01f86ce5978039e7135189d4fecd18c841b31bf2587f6607bcbe9
a76dc112aa74b8b3c058c6d99002a25a6c8157d986b8ad4b3aa187c8f83e6a9181112d8e3dab14599c31bf5f6af8548c4e7eea64b6b50d84a5eb43569df81a399592976d339dbee29ddcf4ec08fc328a49e48147f55
8e5275f14b4bb6",
        "e2cfbf18961b70a5bce94bb50c06af62ba75e6bf6d570b03373833c7bcb6508a168f066dcf511e1e6656dc40ebb013fda1434e79fb8a3f9537e7e4f41ac31259174288696f5643b538579f8c8dc1db80fa
9f6eeb670bf8eb0376eae5345dbd4b4451b1420646e28efa3af2bdf76b0e9ea76e7372d46b9edcf72c098b24cb7eb4d594700ac7f11b6d76b2704225b87408b7f941b6e625cd6aab8cd4c1801483a22155fb8d96028
10f1ea1004694f9dc9c078325ae1a20d7ee8ba25a5712a3f6cd298a45bdf0fd859f6a9f6412e69a1c420d8d60d6da7de3c5b1e9bab38dab73b17a094c39f0e428515b7869fd6f4e872546ac9b056b1459722c02ad9d
0e239e58ec8dfefd9b0fa4b84df10fb7a313832b6f4539cfdb50a9d0b4e0ad2c9dfe309ffc474e3c102865bebdad97c87a2f503342b5e3cdf26a218c68132af66c51465b2d7108292f21652d7e014d377141649161c
74c8e694ae50deeb765d4fc6aaf2e11f2aef7cd4dea78e9b415f8215c580fb5a792c9210eb3c6214042274d47d68c4cedcb72b9fbb027d1ce564ff8bd7606a969c6cc3b608e37b48babec4f11a89601ade0912ee6fa
8e24eb074a64585c0c4d4bae60ff1b14c903cea1d8a8e404dde9d247872b29b9a98d7c67fa4f49d10627279d9a4e0a4ba34ef03c1ee536953ab70c0630186547884d0b173b5f3fffff33786f823c2c804cbd35b1925
f85da075dff522c1ade759c61483ca7321becd16de97084d70846072118004f9d6dc1d514844e5d36b16e008082adec75fb483d6c66b26c8ff6837dc9d86d2bc14ee6955c80610100034dff2ba70e01de052278e165
969e2c1bac88d6"
      ]
    }
  ]
}
```


As mentioned previously, shielded transactions can be of 3 types. The json transaction shown above is a shielded transaction sending shielded inputs to unshielded outputs. Observe that array of vin objects is empty but there is 2 vout objects as well as a vjoinsplit

VJoinSplit

VJoinSplit objects describe the movement of shielded funds. Like unshielded transactions where there many be many vins and many vouts, a shielded transaction may have many vjoinsplits. A JoinSplit will always have exactly 2 coin inputs and 2 coin outputs.

The fields of a JoinSplit are as follows:

- **Nullifiers:** Each JoinSplit will have 2 nullifier values which correspond to the opening values to two previous coin commitments.
- **Commitments:** Each JoinSplit will have 2 commitments which are the coin commitments for the 2 outputs coins of that JoinSplit.
- **Proof:** Each JoinSplit contains one proof, which is the representation of the zk-SNARK zero knowledge proof which proves to any transaction verifier that the transaction is valid without spoiling the anonymity of the transaction sender or receiver.
- **One time Public Key:** this is a one time ephemeral curve Curve25519 public key used to perform in band symmetric key agreement with the receiver of funds.
- **Random Seed:** this is the random seed value for the new coins, simply seeding the coin commitments randomly for correct security.
- **Ciphertexts:** These are the 2 encryptions of the outputs of this transaction. i.e. the encryptions of the revealing values of the outputted coins. They are encrypted under a shared key computed using the above one time public key
- **Macs:** Message authentication codes for the above encryptions

These values facilitate the sending of funds from one address to another without revealing those addresses.

The Zcash Node software

The official Zcash node software is a large and cumbersome program which is difficult to install. Due to it's age there is not a lot of documentation available yet. It time consuming to both install the program and to interpret which commands were needed for our task.

Only minimal install instructions are provided [4] and only for Debian Linux.

The zcash node software suite consists of two programs which we utilise:

- zcashd
- zcash-cli

zcashd is essentially the zcash node software and needs to run for some time in order to retrieve and verify all the blocks and transactions on the blockchain.

One configuration change is needed to the zcashd config file:

The config file is usually located at `/home/<user>/.zcash/zcash.conf` and requires the following line to be added:

```
txindex=1
```

This sets the node to fully index all of the transactions so that they can be queried properly.

After successfully installing, we came upon the two commands we needed to obtain the appropriate data.

```
zcash-cli getrawtransaction <transaction_id>
```

This command returns the raw transaction data from the provided transaction id

```
zcash-cli decoderawtransaction <raw_transaction>
```

This command returns a human readable version of the provided raw transaction

Mining Zcash data

Our strategy for mining the data was to run the full zcash node and query the node for transaction data. We wrote a python program which queries the node through zcash-cli and organises the data in a well structured database.

While we mined all the data from the blockchain, including the data from shielded transactions, we only considered unshielded transactions in our analysis.

The repository for my data mining software is here: `https://github.com/KillianDavitt/ZcashDataMiner`

Signatures and checking for randoms

The zcash unshielded ECDSA signatures were decoded with significant guidance from [5] the similarity between bitcoin and unshielded zcash transactions means that the information provided is quite applicable.

```
CREATE TABLE vin_script (
        script_id INTEGER NOT NULL,
        vin_id INTEGER NOT NULL,
        asm VARCHAR,
        hex_script VARCHAR,
        req_sigs INTEGER,
        type_script VARCHAR,
        transaction_id INTEGER NOT NULL,
        PRIMARY KEY (script_id),
        FOREIGN KEY(vin_id) REFERENCES vin (vin_id),
        FOREIGN KEY(transaction_id) REFERENCES "transaction" (tx_id)
);
```

The ECDSA signature (R,S) is contained in the "asm" attribute, inside the "`vin_script`" table.

The signature is DER encoded. It has a special structure, as follows:

```
3045022100d52330113ccd033ccb1aaa3b759e9696c216e802922e5f1902cd5ada69c612e50220
57880205319dccb05eebbe34323a852ee82653f09f81253ddccd08a810e9d42d01
```

'30' indicates that a compound structure starts there. As a signature is composed of 2 parts, R and S, it is considered as a compound. '45' is the total length of the compound structure. After that, the contents of the compound starts.

Then, it is the same procedure for S. '02' indicates that S is an integer.

'20' is the byte-length in hexadecimal of S, i.e. 32 bytes in decimal.

'57880205319dccb05eebbe34323a852ee82653f09f81253ddccd08a810e9d42d' is S.

'01' is not part of the signature. It is added by Zcash and indicates what fields of the transaction are signed. 01 means "all".

By parsing "asm" for every transaction, we built an SQL table containing the fetched values of R and the associated transaction id. We can see in Figure 1 that $\text{transaction}_{\text{id}}$ is the 10th attribute of the $\text{vin}_{\text{script}}$ table.

To output the duplicate randoms, we used the SQL query:

```
1 SELECT r, COUNT(*) c FROM signature GROUP BY r HAVING c > 1;
```

Even with 20 million signatures as input, no duplicate randoms were printed out.

Why didn't it work & Some stats

Zcash was initially launched on 28 October 2016 which could be considered to be quite recent. Obviously the newer a cryptocurrency is the less transactions there are. Zcash also has significantly less users than larger currencies like Bitcoin and Ethereum. Both of these facts mean that there is much less opportunity to find repeated randoms or other cryptographic anomalies

	Shielded	Unshielded
Number of Transactions	309,258	2,500,00
Number of spends	346,114	39,834,064

13% of transactions are shielded

0.861% of spends are shielded

Reuse of Public Keys

Upon suggestion of Dr. Courtois, we changed our focus to attempt to collected reused public keys. In bitcoin, it is highly discouraged from using the same public key (address) to receive multiple transactions. Doing so has significant risks to anonymity and security [6, 7]

Since Zcash's unshielded transactions behave similarly to bitcoin transactions we attempted to search for reused public keys in the zcash blockchain. This process did not require significant extra effort as all the data needed to determine reused public keys was already in our database.

Above, in our discussion of signatures in vin scripts. We determine how the signature values are contained in the first half of the asm of the vin script value. The second half of the vin script asm, contains the public key as can be seen below

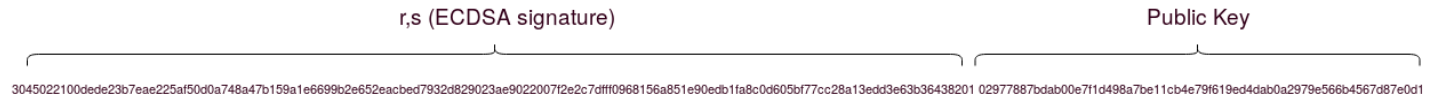

Figure 5: The format of zcash asm data

A simple python script was required to extract the public key information from our transaction database.

`https://github.com/KillianDavitt/ZcashDataMiner/blob/master/pubkey_from_asm.py`

After the public keys and transaction id pairs were collected in a file. Another simple script was required to collate them, remove duplicate transaction id's and count the actual repetitions of user public keys. The following script did this.

`https://github.com/KillianDavitt/ZcashDataMiner/blob/master/count_dupes.py`

Results of repeated public keys

The collection of repeated user public keys proved to be much more successful than the collection of repeated signature random numbers. In total, approximately 20 million signatures used public keys which had already been used. It is immediately clear that reuse of public keys is completely widespread in the Zcash blockchain.

The most reused public key:

035556ffd87eda7df56cf4a1f180022e22572920d70994cb9139ba973f2a105d71

This public key was used to sign 41,138 transactions. This is an enormous number and this account obviously contains a large concentration of Zcash activity. We derived the account address to be:

t1VpYecBW4UudbGcy4ufh61eWxQCoFaUrPs and it can be viewed here:

`https://explorer.zcha.in/accounts/t1VpYecBW4UudbGcy4ufh61eWxQCoFaUrPs`

This particular blockchain explorer has identitied this as a mining pool account "flypool" and this explains why there are so many transactions going through this account.

It is interesting to note that there appears to be no advice relating to the reuse of public keys provided by the Zcash team. And indeed, the official Zcash client is also ambiguous and does not encourage the use of multiple public keys. It is possible that this is a large factor in the reason why so many public keys are reused.

Other potential Cryptographic events

The main focus of this project was to attempt to find repeated random values in ECDSA signatures, however there are a few more potential areas of cryptographic interest particularly in the JoinSplit objects. As is discussed earlier, each JoinSplit includes a one time public key as well as a random seed and 2 ciphertexts. Of course, since the random seeds are public they could be examined for collisions. The one time public keys could also be examined for collisions.

As of today, we don't believe that there exists enough shielded transactions on the blockchain to make these examinations worthwhile. In fact we checked for collisions of the "random seed" values and of the 346,114 checked, there were no duplicates.

One would imagine that if our software was run again in the future after the Zcash blockchain grows, there would be a much higher chance of finding both the unshielded repeated randoms and possibly more interesting repitions in the VJoinSplit object.

Conclusion

In conclusion, our search for duplicated random values in signatures was not sucessful. However, many many duplicates public keys were found. While it appears the designers of Zcash were careful to avoid any problems with repeating randoms, they did not properly discourage reuse of public keys.

We also suppose that one possible reason that no random collisions were found is the uniformity of clients. Currently most users of Zcash are encouraged to use the official client and there does not exist a huge number of alternative clients as there does with other coins like bitcoin. However as Zcash grows there may be potential for more errors in randoms to occur. In addition, as Zcash grows, it should be beholden to the Zcash team to discourage more strongly, the reuse of public keys.

References

[1] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*, pp. 459–474, IEEE, 2014.

[2] Zcash Team, "Zcash." `https://z.cash/`, 2017. [Online; accessed 22-February-2017].

[3] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol." `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf`. Accessed: 2018-03-13.

[4] "Zcash user guide." `https://github.com/zcash/zcash/wiki/1.0-User-Guide`.

[5] N. Schneider, "Recovering bitcoin private keys using weak signatures from the blockchain." `https://www.nilsschneider.net/2013/01/28/recovering-bitcoin-private-keys.html`. Accessed: 2018-03-13.

[6] F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Security and privacy in social networks*, pp. 197–223, Springer, 2013.

[7] "Address resuse: bitcoin wiki." `https://en.bitcoin.it/wiki/Address_reuse`. Accessed: 2018-05-04.